Tagging Design

Goal

The CMR should support "tagging" of collections to allow users to group related collections together in a way that is queryable.

See the Tagging Concept Of Operations for a description of tagging

Traceability

CMR-1952 - JIRA project doesn't exist or you don't have permission to view it.

Design Drivers

- Support the requirements as outlined in Jama
- Implement this by the end of September.
    - Choose simple solutions. Reuse code and existing capabilities where it makes sense.
- Keep Metadata DB storage of tags in sync with whatever is indexed in Elasticsearch.

Design Overview

- API
    - The API for CRUD of tags is hosted in the Search application
- Storage
    - Tags and a list of the associated collections are stored as a single concept type in Metadata DB
    - They are revisioned.
- Indexing
    - Tags with the associated collections are stored in a new index in elasticsearch

Tag API

This will be hosted on the Search API and inside the search application.

Justification: Tags are really all about search. Any user can tag sets of collections as a way to group them for later searching. There's an argument to make that tags should be in a new microservice and a new top level URL /tags. There's enough overhead in doing this that we can't justify it with the current deadline. I'm not sure that it's actually something we'd want to do anyway.

/tags

POST - Create a tag

Requires a token to be sent in the header which is used as the originator ID.

JSON body

```
{
  "namespace": "org.nasa.something",
  "category": "QA", // optional
  "value": "quality",
  "description": "A very good tag",
}
```

Response will contain the tag id and the revision id of the tag.

GET - Find tags by combinations of parameters

namespace, category, value, originator_id are all supported. Retrieval of a tag should be in JSON and should include the POSTed fields along with tag-id and originator id.

Search results will look something like:

```
{
"hits": 124,
"took": 1234,
"tags": [{
"tag-id": "T12345",
"namespace": "org.nasa.something",
"category": "...",
"value": "quality",
"description": "A very good tag",
"originator-id": "jgilman"
}, {
"tag-id": "T12345",
"namespace": "org.nasa.something",
"category": "...",
"value": "quality",
"description": "A very good tag",
"originator-id": "jgilman"
}]
}
```

/tags/:tag-id

tag id is equivalent to concept id.

PUT - Update a Tag

DELETE - Remove a Tag

/tag/:tag-id/associations

POST - Associate this tag with collections

The body must be a JSON query. All collections found will be added to the associations on the tag.

Steps to modify a tags associations

1. Find latest tag in metadata db by tag id
    a. Return 404 if no tag by tag id or if it's deleted.
2. Search for collection concept ids using the JSON query.
    a. Use a :unlimited page-size and select out just concept-id.
3. Add the concept ids to the :associated-concept-ids of the tag
4. Associate the current user-id with the tag being saved. (Goes in concept map. The originator id should not be replaced.)
5. Save the tag back to metadata db with an incremented revision-id

a. Metadata DB will reject the update if a concurrent request was received. We will let this error be returned back to the user. This needs to be explicitly tested. It shouldn't result in a 500 error.

DELETE - Disassociate collections with this tag.

The body must be a JSON query. All collections found will be removed from associations with the tag. We'll ignore any collections that aren't already associated with the tag.

Tag Validation

- A valid user token is required for any tag actions. This is used for the originator id.
- Use JSON schema validation on creation or update. This will handle valid JSON, required fields, types, and sizes.
    - Namespace and value are required
    - namespace string 1 - 514 characters (This is half of native id max size, 1030 - 1 for a separator)
        - Can be any character except separator character
    - value string 1 - 515 characters
        - Can be any character except separator character
    - category string 1 - 1030 characters
    - description string 1 - 4000 characters
- Create specific validations
    - Namespace and value must be unique.
- Update validations
    - Namespace and value can't change.
        - This is used for the native id in metadata db.
- Delete validations
    - Tag must exist.
- Association and Dissociation validations
    - Query must be valid.
    - Tag must exist and not be deleted

Tag Storage

Tags will be stored in Metadata DB. Immutability and use of revisions has worked well for collections and granules. It allows us to easily keep Metadata DB and Elastic in sync. It has some overhead but we've figured out how to address many of the challenges of multiple revision.

Tags as Concepts

Tags will be stored as concepts in metadata db. This would allow us to avoid adding much code to metadata db.

Benefits of this approach:

- Defined storage APIs that just work
- Defined search APIs that just work
- Revisions and associated creation and cleanup
- Id generation
- Existing client libraries that would also just work.
- Any code paths that handle concept maps would also likely work.

Issues with this approach:

- Providers do not own tags. Provider id is one of the required fields of concepts
- Tags don't have a single native id.
- Tags aren't really concepts.

Alternative approaches

- New Tag specific API on Metadata DB
    - We could store tags in their own database table and define a specific API and processing code for them. This would be a lot of

work and would add code very similar to what we've already added. We know we need revision storage for tags.
- New Versioned Data API on Metadata DB
    - This would be a new API similar to the concept API but not owned by a provider and used for things that are not concepts. This isn't a bad approach but it requires duplicating many of the things that are already implemented in Metadata DB. It would be a lot of work. Also, would there be anything besides tags that we'd need to store this way?

Tags as Concepts Design

System Level Concepts

Tags are not associated with a provider. We will introduce the idea system level concepts to represent data that is not owned by a provider. A provider id of "cmr" will be used to indicate it's owned at the system level. Within metadata db when receiving concept maps we should convert any provider-id of "cmr" to a keyword :cmr to indicate it's a special case. Existing code paths assume you can take a provider id, retrieve the provider, and then use that on various cmr.metadata-db.data.concepts protocol functions. We could add support for :cmr provider id that would allow retrieval of the system provider that could be passed to all functions.

Tag Namespace and Value are Used as Native Id

Namespace and value would be combined to use as the native id of a concept. Native ids can be up to 1030 characters long. That would give 514 characters for namespace, 1 for a separator character, and 515 for the value. The separator character can be one of the non-printable ascii characters like Group Separator (char 29). See http://www.asciitable.com/. We would not allow namespace and value to contain this character. We couldn't use a common separator character like "/" because we would want to permit namespaces and values to contain that and we would need to escape it. Escaping the characters would require additional space in the database and reduce the number of characters available for namespace and value.

Tag Concept Ids

The tag concept id will be of the format "T<sequence-num>". Concept ids normally have a "-<provider-id>" in them but because tags are owned at the system level they will not contain that. We ill need to modify our concept id parsing code to return :cmr as the provider id when parsing a concept id without the dash provider id.

Tag Representation as a Concept Map

```
{:concept-type :tag
 :native-id "the namespace\u001dthe tag value"
 :concept-id "T1"
 :provider-id :cmr
 :user-id "user101"
 :metadata "..." ;; edn as described below
 :format "application/edn"
 :revision-id" 1
 :revision-date" "2012-01-01T00:00:00"
 ;; TODO determine extra fields for tags. What do we need to search on or ensure uniqueness?
 :extra-fields" {:tag-namespace "..."
                 :value "..."}}
```

Tag Metadata

```
{:tag-namespace "org.nasa.something"
 :category "..."
 :value "quality"
 :description "A very good tag"
 :originator-id "jgilman"
 ;; Using EDN here allows us to make this a set.
 :associated-concept-ids #{"C1-PROV1" "C2-PROV2"}

;;This is an idea we could use for CMR-1894 for tagging a specific revision of a collection

:associated-concept-revision-ids #{["C1-PROV1" 2]}}
```

Tag Indexing

Indexing of tags will be handled by the indexer. The indexer currently handle events of updated concepts from ingest. Tag events will be similar to the existing ones. We can either have the Search app send these events when tags are changed or move the production of events from Ingest to metadata db. This would be one place for producing these events regardless of where the original data comes from.

We will need to add a new tag index to the index set created by the indexer. We should store all fields in a tag. Associated collection ids should be stored in elastic in a field that is not indexed (index= "no"). The set of collection ids should be stored as base 64 encoded, gzipped, EDN of the set.

Searching for Collections by Tag Fields

Related Item Queries

We need to be able to search for collection by fields that are indexed with tags. We will need to execute a query for tags fetching the associated collection concept ids and use them to find collections with the other conditions. We could represent the concept of finding related items in order to find something else as a generic concept. We would introduce a related item query condition

```
(defrecord RelatedItemQueryCondition
[
concept-type
condition
result-field
matching-field
])
```

The fields would have the following meanings

- concept-type - What item should be found? In this case the concept-type would be :tag
- condition - The condition for finding related items. If tag-namespace was the parameter then this would be a string condition to find tags where the namespace equaled a certain value.
- result-field - What field should be retrieved when finding the items. This would be set to :associated-concept-ids when finding tags.
- matching-field - The generated condition after executing a related item query would want to match the result field to some other fields in the item the overall query is finding. When finding tags by collection this would be set to :concept-id.

The related item query would be useful for finding granules by collection fields and in the future for finding other kinds of concepts like documentation and parameters by related item fields.

Not Using This Approach

The related item query approach is appealing for the reasons listed above but there are a few reasons we'll wait to implement it now.

- It would be difficult to implement. The concept is flexible but somewhat complex. There are simpler faster ways to implement this in the time available.
- The code related to finding granules by collection parameters is performance sensitive. We'd want to avoid executing a separate query for each related item query condition. They need to be grouped together. This is a little tricky to do.

We can revisit this approach in the future if there's new needs for it.

Translating Tag Parameters to Collection Concept Ids

An alternative easier approach that we will use is to extract tag parameters from a collection query during parameter processing. We'll take all of them, query for tags and then replace with a set of collection ids. If tags don't match any thing the concept id would be set to NO_MATCH or something like that.

Error rendering macro 'pageapproval' : null